DMD-II Assignment 1

Prepared by Daniil Manakovskiy
d.manakovskiy@innopolis,university
B18-02

Requirements

- Docker
- Docker-compose
- Python3

File structure

```
.
├── docker-compose.yml
├── misc
│   ├── component.plantuml
│   ├── component_diag.png
├── neo4j
│   ├── data
│   ├── logs
│   └── plugins
│       └── apoc-3.5.0.9-all.jar - a plugin to parse datetime from string
├── pgbackup - directory with postgres backup
├── queries
│   ├── neo4j
│   │   ├── query{i}.py for i in [1..5] - neo4j queries
│   │   └── Query2_formatted.xlsx - an example of formatted result of Q2.
│   └── postgres
│       └── query{i}.sql for i in [1..4] - corresponding sql queries
│           (they aren't required, but let it be)
└── src
    ├── migrate.py
    └── requirements.txt
```

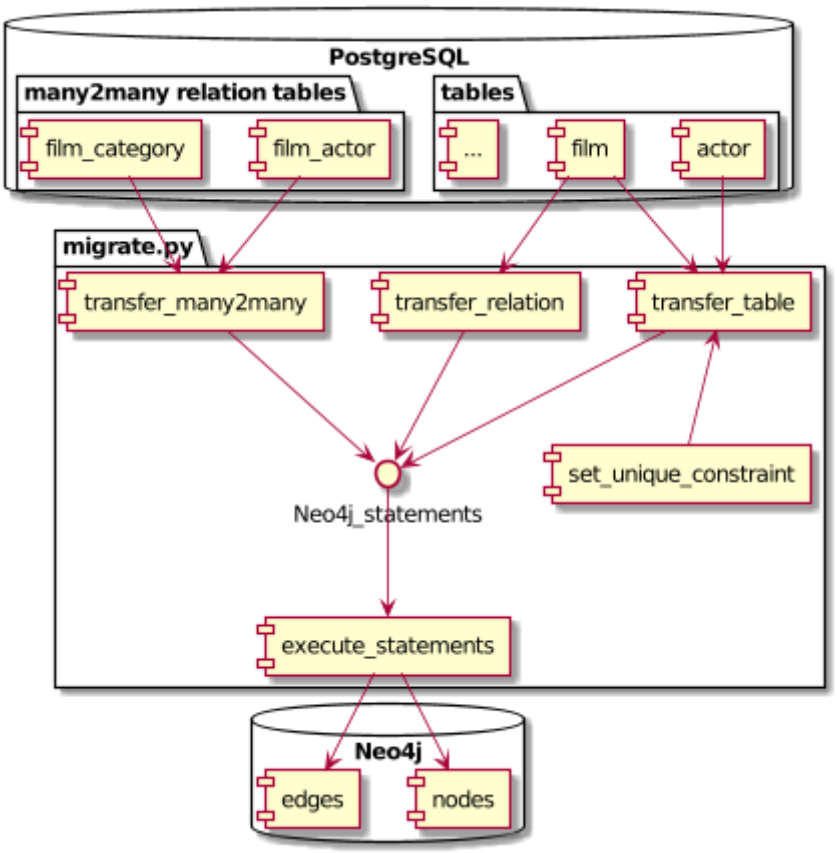
How to run

0. go to an assignment folder
1. execute `pip3 install -r src/requirements.txt`
2. run `docker-compose up` (be sure ports `5432`, `7474`, `7473`, `7687` are available)
3. wait until the system fully start up

4. execute the following command to start restoring
   `docker exec -it dmd2A1_postgres psql -U DMD2user -h localhost -p 5432 -d dvdrental -f /backup/restore.sql`
5. execute migration script: `python3 src/migrate.py`
6. execute queries: `python3 queries/query{#}.py` , where `{#}` is a number of wanted query

Moving the database

A component diagram for `migrate.py` is shown on a figure below



Brief explanations of functions:
all functions below export data from postgres in csv format

- `transfer_table` - given `table_name` return a tuple of neo4j queries for creating unique constaint on id and import the csv of table
- `transfer_relation` - returns a neo4j query to import edges from csv of relations between 2 tables
- `transfer_many2many` - returns a neo4j query to import edges given in many-to-many relationship table
- `execute_statements` - executes given list of statements, supressing `neo4j.exceptions.ClientError`

Adjustments made

1. Neo4j does not support `decimal` datatype, thus all such fields were converted to `float`
2. Since there is no constraints on number of edges between nodes, backend must validate all one-to-one relations.
3. Due to implementation of `neo4j` python library, all `datetime` fields were converted to `long` and contain timestamps. For some reasons driver does not execute import statements having nested functions call like `datetime({epochmillis:toInteger(apoc.date.parse(row.last_update))})` without any errors, meanwhile in neo4j browser such queries are being executed OK.

Performance

query time execution for Postgres were measured by DataGrip,
for neo4j by python `time` library

Neo4j has b-tree indices on all properties ending with `'_id'`

Task	Postgres	Neo4j	Neo4j speed advantage
DB migration	-	~27 sec	-
Query 1	0.1 sec	0.1 sec	0%
Query 2	0.4 sec	0.68 sec + 2.6 sec for representation	-70%
Query 3	0.18 sec	0.09 sec	+100%
Query 4	0.27 sec	0.12 sec	+125%
Query 5	was not measured	0.17 sec	?

Result: Neo4j performes

- equally on queriyes involving cross product of 2 tables directly,
- worse on cross product of many-to-many relations (Q2) since it have to find all paths between 3 nodes via 2 relations (~ 2 joins) instead of doing 1.
- twice as much better on queries requiring to find long path between nodes.